A Brief Exploration of Image Search Methods

Elliot Pickens
Cornell Tech
eep58@cornell.edu

Abstract

In this paper I approach the question "how can we return the best matching image to a natural language query." I tried to answer that question through the lens of machine learning by starting from a baseline regression model, and then building up up to modern deep learning methods. I ultimately found that using CLIP (Contrastive Language-Image Pre-Training) produced the best results.

1 Data

For this project I was given a rich data set containing 12,000 images split between training and test sets. Each imagine was accompanied by a set of descriptions, tags, and extracted ResNet features. There were approximately five descriptions and tags per image, but a small number had fewer than five. The ResNet features were taken from the third to last layer of the model after being fine tuned on the data and were in the form of a 1×1000 vector.

2 Working With the Baseline

We were initially given a baseline model that used a 300 dimensional sentence embedding produced by `word2vec` as the input data to a ridge regression method that was set to predict a 100 randomly selected ResNet features. Despite the inherent stochastic-ity present in this method it actually produced results that we not all that bad considering the difficultly of the problem. This method was able to hit a MAPE score over 0.12. After inspecting this baseline I determined that before jumping into more complicated approaches I should apply some minor modifications to the baseline.

2.1 Initial Tuning

The first of those modifications was swapping out ridge regression for a elastic net regression. I did this to allow for a few additional tune-able parameters. The most important of which was the ability to add some additional regularization by tuning the L1 and L2 ratio as well as the associated alpha value. I wanted to add regularization early on to see it would help handle the high dimensional nature of the data. It made a difference, increasing the performance of the model by a small amount, but it was not enough to make a dramatic impact.

2.2 Boosting and PCA

The next step I took to improve upon the baseline was to remove the randomness caused by the random ResNet feature reduction. To do this I replaced the randomized selection with PCA. I found that PCA improved performance moderately.

At this point decided that it was time to move on from the initial regression model at the core of this approach. I turned to XGBoost. XGBoost is a fast an efficient boosting algorithm that has a strong

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

Early Results	
Model	Test (Kaggle) Score
ElasticNet	0.127
ElasticNet + PCA (n=100)	0.143
XGBoost + PCA (n=100)	**0.173**

track record of performance across a wide range of tasks. [1] I tested XGBoost on both the random and PCA reductions of the ResNet features. Both had showed a reasonable boost in performance.

3 Diving Into Deep Learning

Although, I could have spent more time working with the initial boosting method and increasing its performance I decided to take things in a different direction and employ a series of neural network based models. My first attempt at this was to replace the regression model with a neural network setup to do regression implemented in `Pytorch`. Sadly, this failed miserably. I attempted a number of different approaches for this. First off was the simple neural network mapping description embeddings directly to ResNet features. Of course, this introduced a very problematic scenario where I was trying to fit a model $f(X) = \hat{Y} \approx Y$ where X is a $n \times p$ matrix and Y is a $n \times m$ matrix where $m \approx 3p$. Under such conditions it is difficult build a functioning model, so I quickly reduced the size of the ResNet features using the PCA method described in the previous section as well as an autoencoder method. I assumed the PCA method would result in a working model at the very least, but I was much less certain about the autoencoder approach. Sadly, neither worked and they both registered very poor performance.

Why did this happen? A few different reasons come to mind. First off is the model. I tried a number of different configurations for the neural network I used to predict ResNet features from `word2vec` encodings. I tried both increasing and decreasing its size as well as introducing various levels of regularization through dropout and weight decay. I ended up with more or less the same result regardless. These performance issues came as quite a shock. I was not doing anything particularly differently than I had done during my initial exploration, and yet the results varied wildly from the previous results. My suspicion is that even with added dropout layers and weight decay the model was still over-fitting to the training data. I believe this to be the case due to the fact that the loss during the training process (MSE) was very low and approached zero quite quickly, but MAPE on the validation and test sets was nearly 0.

Following these results I moved to swap out the previous feature reduction via PCA with an autoencoder method. In retrospect moving to add an autoencoder my pipeline at this stage (having seen poor results using PCA and direct mapping) was not a great idea, and one made out of confusion more than anything but in the moment it made more sense. I tried training an autoencoder on the ResNet features in order to find what would hopefully be a better way doing dimensionality reduction than PCA. It was not. The autoencoder was unable to effective encode and decode the ResNet features with any of the configurations I tested. Given the efficacy of PCA in doing decomposition I believe this can be attributed to human error on my part. It was a good practice exercise, but not one that improved the model.

All in all I don't have a conclusive answer as to why these early deep neural network models did not play out the way I expected. Had I had more time to run tests I think I would have figured out the flaw in my approach, but with limited computational power I chose to cut my losses early and quickly move onto my next approach.

3.1 Image Captioning

Despite these difficulties I did not choose to completely give up on incorporating some sort of deep learning approach into the project. Rather than use a neural network as my model I shifted to consider how I might be able to augment the data I had through existing pre-trained neural networks. Given the recent rapid advances in natural language processing on vision language tasks it seemed like it could

[1] It also happens to be a favorite among Kaggle competition participants, so I thought it was only right to use it as part of this project.

be a perfect match. If it would be possible to correctly caption the images in my data set then the core then search performance could be drastically improved. Through this approach I would process every image with a pre-trained network and then store every output caption. With those captions saved I would then search for images using cosine similarity between description embeddings and caption embeddings.

To do this I used `HuggingFace's` vision language tools to put together a network that would take in an image and output a caption. I employed the Vision Transformer (ViT) model as the image encoder and GPT2 as the text decoder for this task. For this approach an image would be sent through the encoder, which would output a sequence that the decoder would then decode as a caption.

Figure 1: Two Cats Captioned: "I don't know if I'm going to be"

The model was able to caption images, but none of the captions were very good. One example image caption pair can be seen in figure 3.1 above. Clearly it is not correct, and not anywhere close to correct at that. I was surprised at just how badly the captions I got were, but not completely shocked. It is not advised that pre-trained models be used directly on specific tasks and that showed to be very true in this case. I had hoped that I would be able to properly tune this model locally on my machine, but that proved to be too computationally expensive so I was unable to produce captions that were worthy of use in my model.

4 CLIP

Having faced a solid number of discouraging results during my excursion into deep learning I was almost ready to give up and return to the boosting method previously mentioned, but before quitting on the methodology I gave it one last shot by turning to CLIP (Contrastive Language-Image Pre-Training). CLIP has shown great performance on a number of tasks using an approach that directly ties image and language data. The key to CLIP's performance is its training cycle where paired images and text descriptions are encoded and then decoded to train the network.[2] Through this approach CLIP learns to effectively encode both images and text. This works fantastically well and lends itself to effective out of the box use. That single shot capability has the added benefit that it makes for easy search retrival if you have both text and image embeddings. Once those are in hand we can use tirhe dot product to get cosine similarity and then sort and select the best matches.

4.1 Pre-Trained CLIP

To use CLIP on the data provided for this project I used the pre-trained CLIP model provided by the paper authors. Using that model I computed and saved the image embeddings for each image in the train and test sets. I then cycled through each description and computed its text embedding. Each image's text embedding was then dotted with its image embedding and the resulting values were sorted to find the top matches.

The implementation of CLIP provided by the original authors is not set up to quickly be tuned at the moment, so I was not able to fine tune the model for this specific task. That being said, there was still

[2] I do not describe CLIP in great technical detail in this paper, but I might do so in a future blog post.

some room for creativity to maximize the results from the pre-trained model. In particular, I was able to improve upon the most straightforward use of CLIP for image search on the data by modifying the way descriptions were used by the model.

CLIP Results	
Model	Test (Kaggle) Score
Description Averaging	0.628
Full Description	0.684
Punctuation Removed	0.654
+ "This is"	0.689
+ "A photo of"	0.6898
Mixed Start Phrases	**0.697**

I first had to decide how to handle the multiple descriptions that were given for each image. At first I tried taking the sum of the embeddings from each description and dotting the resulting vector with the image encodings. I also tried doing the same thing using the mean of the individual description text encodings. Both methods produced near identical results, which was to be expected given they should be equivalent. After testing these methods I moved on to simply creating one embedding for the entire description. This easily out performed the averaging method, which makes sense given that many transformer based methods (like CLIP) benefit from using all possible context available. Once I had confirmed that a single embedding was best I decided to play around with how I pre-processed the text before encoding it. First, I tried adding, removing, and altering punctuation. None of these alterations improved performance. I then decided to use a method discussed by the CLIP authors: altering the beginning of the descriptions to make more explicit what was shown in the image. The authors mention using phrases like "an image of," "a bad photo of," and "a corrupted jpg of." In their package release, the authors present a more in depth mix of these precursor phrases and describe how they can improve performance. I could not effectively re-train the model so I instead opted to add the most generic of these phrases to each description. I started by adding "this is" to each sentence. It resulted in a moderate bump in performance, so I moved on to tried adding "a photo of" along with a few other starter phrases. I found that "this is" and "a photo of" worked best. I also tried mixing the two and found that to be even more effective. I was able to do this by realizing that each description began with "a..." meaning that a little additional info at the beginning of the sentence wouldn't ruin its meaning.

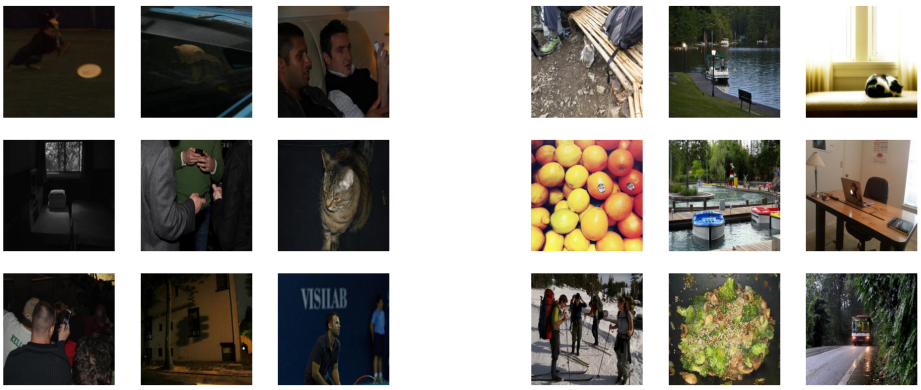
Although, I did not have the time to test anything other methods, I think it would also be interesting to try expanding the available data in other ways. One that comes to mind is using a translation tool to convert the descriptions to another language and then back to English. I know this trick has been leveraged by others for similar tasks, so I seems plausible that it would work in this context

4.2 Baby CLIP

I was very impressed with how well CLIP performed on the data, but I wanted to further investigate the method and learn more about its inner workings by adapting some existing open source implementations of CLIP.

Using the framework of those implementations I trained two "baby" versions of clip on my laptop. Both of these used DistilBERT as the text encoder, but one used ResNet50 as the image encoder while the other used DiET distilled. Neither performed nearly as well as the actual CLIP model, but it was interested to play around with the models and try to make some alterations and observe what happened.

As we can see in the images in 4.2, neither of the "baby" CLIP models on trained on my laptop performed very well. It looks like they're just picking random images from a human perspective, but they did appear to be learn something even if it is not intelligible here. If I were to give a particularly generous interpretation of results I would say that DiET model appears to be retrieving some park images, while the ResNet model returns some pictures of men. I think this is a bit of reach, but there's probably something going on there. If I were to run this again I would try to use an even larger data-set, and run it for more than a single night to test the true performance of my altered version.



(a) ResNet Top 9 (b) ViT Top 9

Figure 2: Top 9 Images for "A man skateboarding in a park"

5 Conclusion

Figure 3: Tweet From @ResNetXtGuesser Showing a Comical Burger Classified as a Pineapple

All in all I think this project showed me both the power of, and the challenges associated with deep learning. I was able to get some results on the test data that I found really quite astonishing, but I was not able to do it using the computational power of my laptop alone. I had to rely on the power of a lab with the budget needed to hire researchers and train massive models like CLIP that I ultimately settled on for this project. Comparing the early regression models, with my initial neural networks, and both real and "baby" CLIP I think it is also possible to see how hard it is to achieve state of the art performance on tasks involving language and vision. I think the meme above encapsulates this perfectly, showing just how hard even a "basic" labeling task can be.

It was a great learning experience and I was able to greatly improve upon my `Pytorch` abilities through all of the trouble shooting needed to get things working. I cannot wait until I get a chance to work on another great project like this!

6 Works Cited

- An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale
- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
- Learning Transferable Visual Models From Natural Language Supervision
  - https://github.com/openai/CLIP
  - https://github.com/moein-shariatnia/OpenAI-CLIP
  - https://github.com/haltakov/natural-language-image-search
- https://huggingface.co/docs/transformers/model_doc/visionencoderdecoder
- Attention Is All You Need
- https://xgboost.readthedocs.io/en/stable/
- https://huggingface.co/docs/transformers/model_doc/vit

Please forgive the informal list of references.